Artifact JSON-RDF Converter

This is a preliminary page intended to trial using the Artifact design methodology (simplified) to guide and describe the project. At this stage it is a brainstorming WIP

References:

1. Central Data Service playground proposal
2. Artifact design methodology (simplified)

Formulating the design problem

Improve (or solve) a ***<problem>***
by designing an ***<artifact>***
that satisfies ***<requirements>***
in order to achieve ***<goal(s)>***

Fundamental components of an artifact design

Problem
Tip: Specific issue or challenge that requires a solution or improvement. Data format incompatibility between data middleware and reasoner

Goal
Tip: Ultimate objective(s) that a solution aims to achieve, typically formed by the stakeholders' desires. In the context of COVESA, the goal of an artifact is inherit from the general COVESA goals defined as an alliance. In other words, each artifact will represent (minor or major) steps towards an ultimate goal. Enabling the data exchange between data middleware and reasoner via websocket

Requirements

Tip: Criteria and specifications that the artifact must meet to address the identified problem and achieve the set goals. Typically presented as functional and non-functional.

- ***Functional**: Specific functions, tasks, or actions that the designed artifact must perform to proof utility.*
- ***Non-functional**: Specific qualities or characteristics that the artifact must have. They represent the constraints under which the design must operate.*

Functional:

- Flexibility in JSON schema:
  - The converter should be able to handle different JSON schemas and adapt to changes in the schema without requiring significant modifications to the code.
- Dynamic mapping:
  - The converter should support dynamic mapping between JSON and RDF, allowing for different mappings based on the specific JSON schema and the desired RDF representation.
- RDF generation:
  - The converter should generate RDF data that adheres to the desired RDF representation, including appropriate subject-predicate-object triples.
- Handling complex data structures:
  - The converter should be able to handle complex JSON data structures, such as nested objects and arrays, and convert them into appropriate RDF representations.
- Error handling and reporting:
  - The converter should handle any errors that occur during the conversion process and provide meaningful error messages or logs to aid in debugging.

Non-functional:

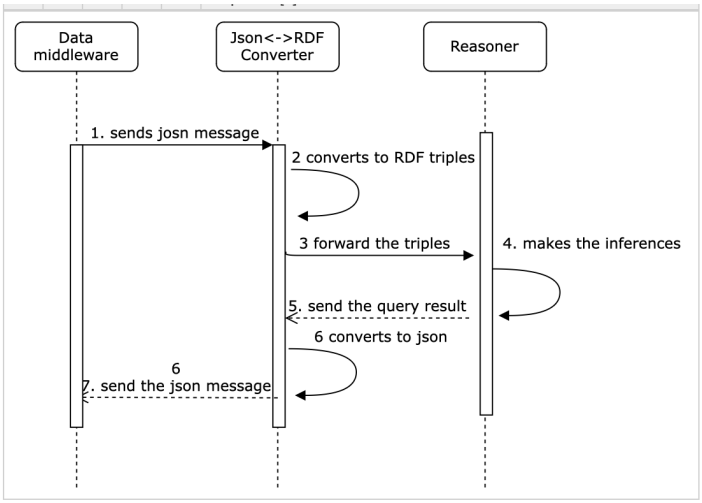
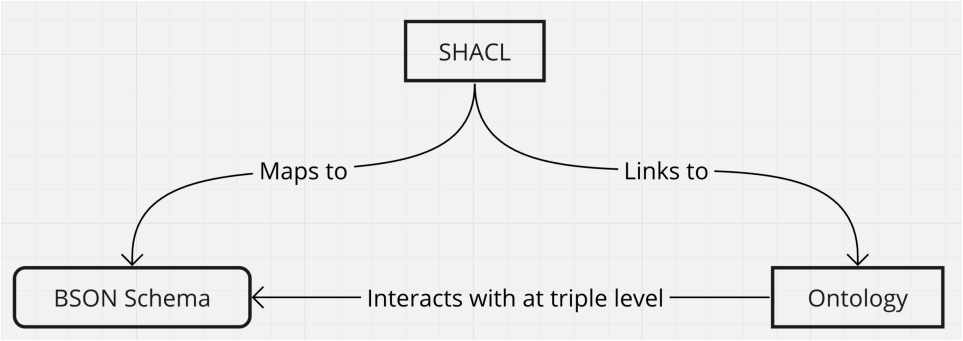
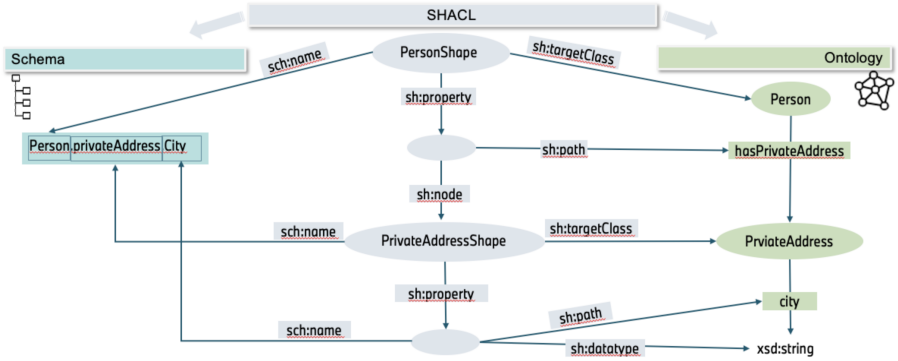
- Reliability:
  - The converter should be reliable and robust, ensuring that data is accurately converted without loss or corruption.
- Maintainability:
  - The converter should be designed in a way that allows for easy maintenance, updates, and bug fixes, ensuring long-term sustainability.
- Usability:
  - The converter should have a user-friendly interface or API, making it easy for developers or administrators to configure, monitor, and interact with the converter.

Artifact

Tip: Represents the tangible outcome of a design that aims to solve the problem and fulfils the specified requirements and goals.

Convertor

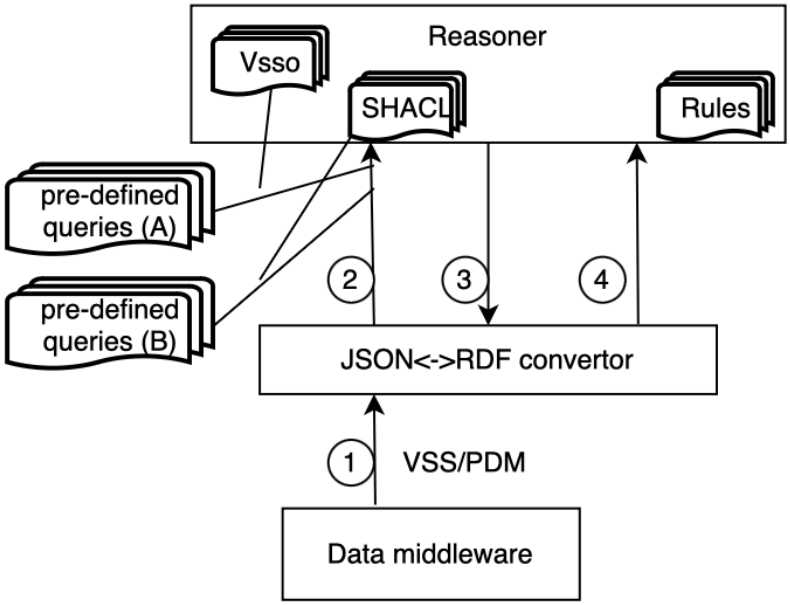
Technical Description:

Sequence Diagram	Provide an overview of the **real-time** converter's architecture, including the components involved and their interactions. This can be depicted using a high-level archit this converter focus one "real-time" side. 
Mapping	Describe the mechanism used for mapping the JSON data to RDF format. Explain how the converter identifies the relevant JSON schema and maps the data elemer illustrate this process.  Example: 

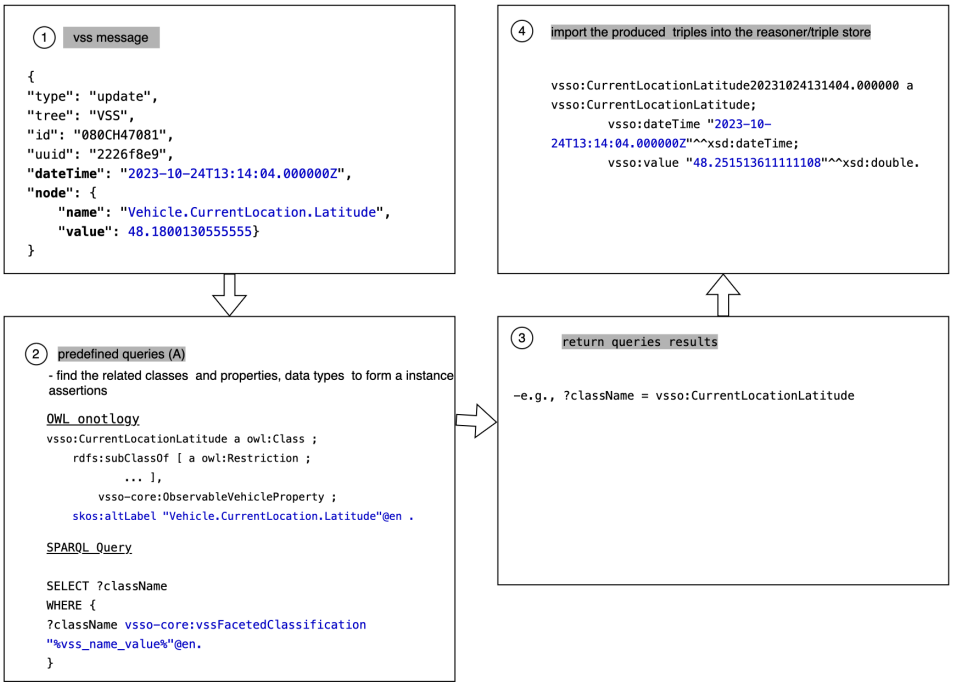
RDF Generation	Explain how the converter generates RDF data from the mapped JSON data. Describe the process of creating subject-predicate-object triples or nodes in RDF forma its corresponding RDF representation.
	
	The process of RDF triple generation are based on the triple categories:
	
	• 1) type assertion:
	  ○ ex:Person1 rdf:type ex:Person.
	• 2) object property assertion:
	  ○ ex:Person1 ex:hasFriend ex:Person2.
	• 3) data property assertion :
	  ○ ex:Person1 ex:age 25
	
	There are two types of messages modelled in different ways which needed to treat differently.
	
	The main difference lies in where does the types, properties and data type come from.
	
	[table below]

Real time JSON to RDF	

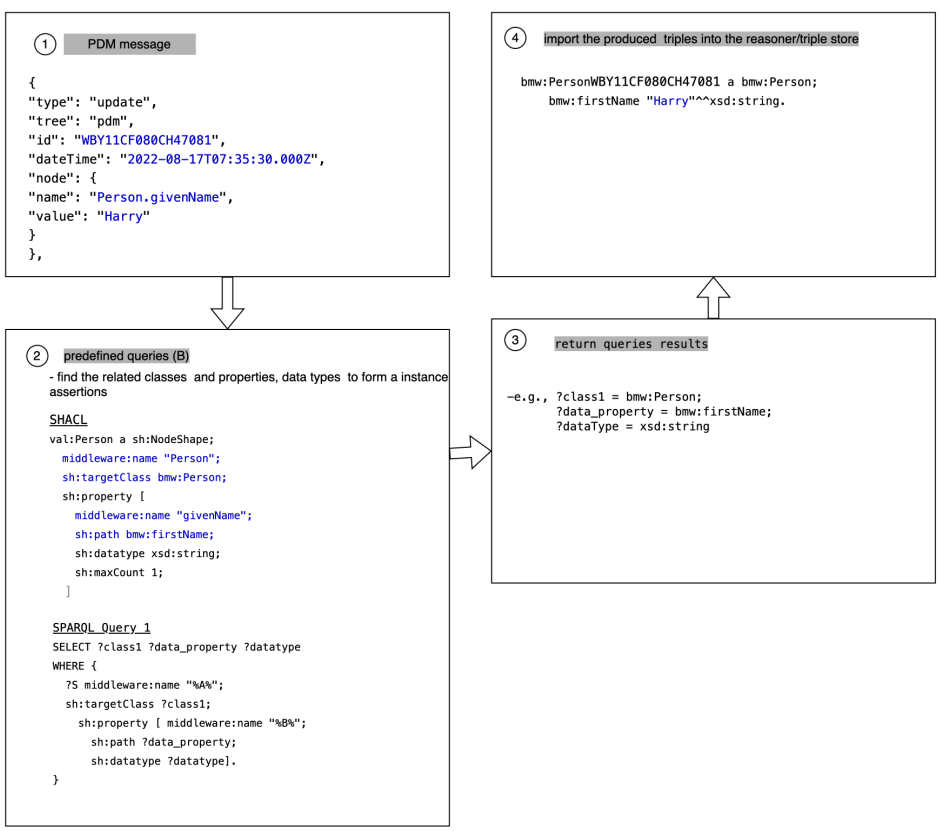
	type assertion	object property assertion	data property assertion
VSS style	VSSo Ontology	VSSo Ontology	vss.json
PDM style	PDMo SHACL	PDMo SHACL	PDMo SHACL



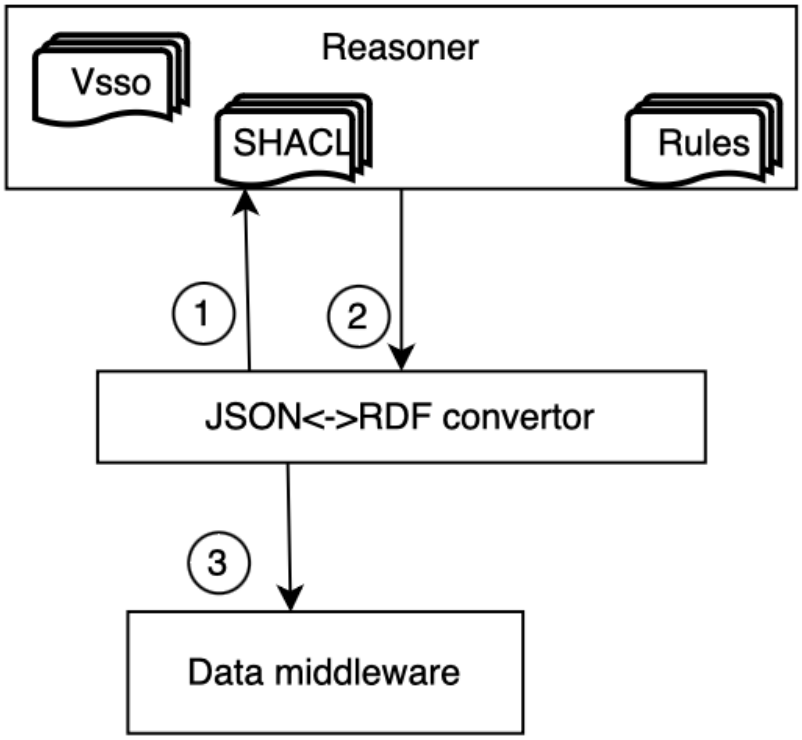
VSS Tree:

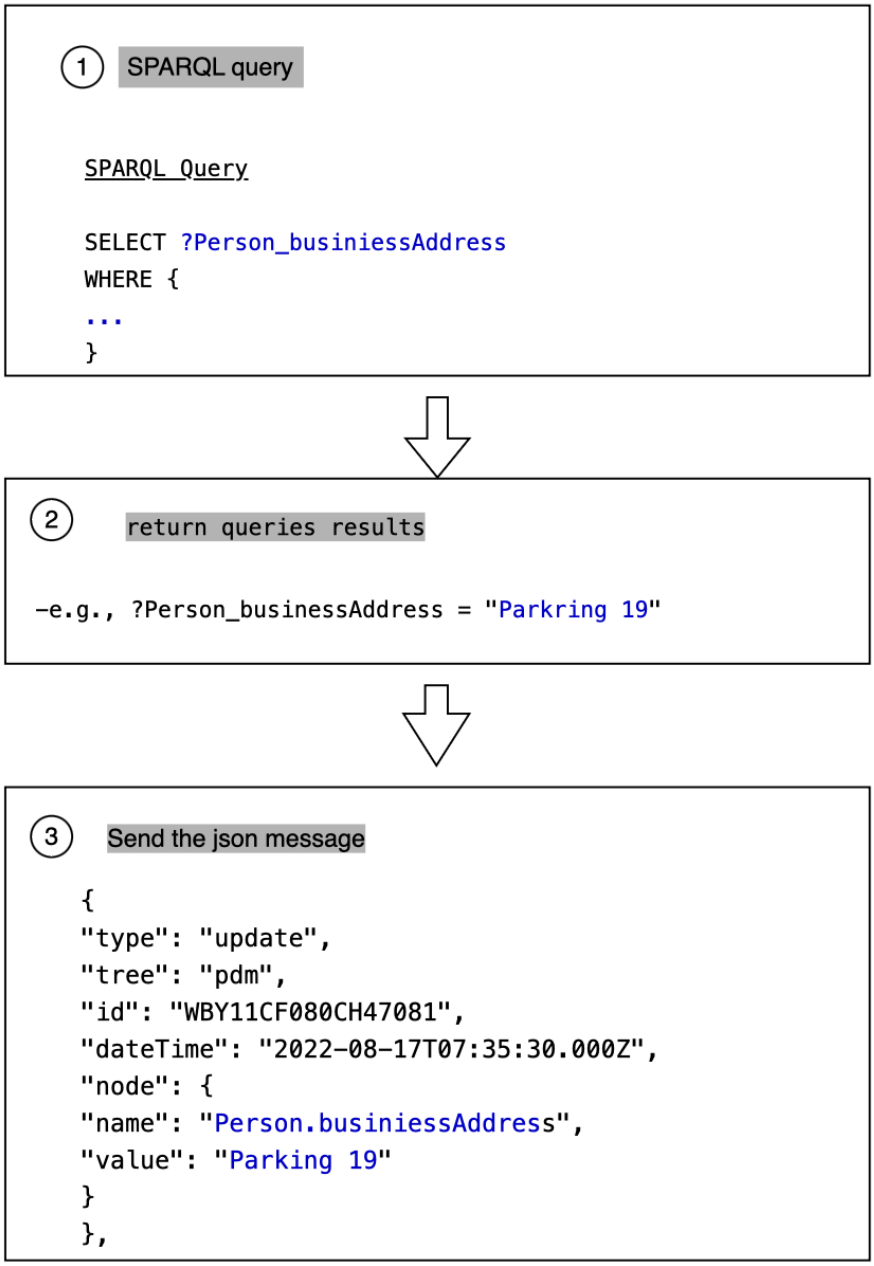


PDM Tree:



Real time RDF to JSON





Integration with Data Middleware and Reasoner	Describe how the converter integrates with the data middleware and the reasoner. Explain any specific APIs, protocols, or connectors used for communication betwe illustrate the integration. • websocket client listens to the middleware ip:port

Configuration and Extensibility	Discuss how the converter is configured and how it can be extended Configuration: Input 1. Tree subscription 2. data points subscription Output 1. data points required